# Distributed, Scalable Routing Based on Vectors of Link States

J.J. Garcia-Luna-Aceves, *Member, IEEE,* and Jochen Behrens, *Student Member, IEEE*

*Abstract*— **Link vector algorithms (LVA) are introduced for the distributed maintenance of routing information in large networks and internets. According to an LVA, each router maintains a subset of the topology that corresponds to adjacent links and those links used by its neighbor routers in their preferred paths to known destinations. Based on that subset of topology information, the router derives its own preferred paths and communicates the corresponding link-state information to its neighbors. An update message contains a vector of updates; each such update specifies a link and its parameters. LVAs can be used for different types of routing. The correctness of LVAs is verified for arbitrary types of routing when correct and deterministic algorithms are used to select preferred paths at each router and each router is able to differentiate old updates from new. LVAs are shown to have better performance than the ideal link-state algorithm based on flooding and the distributed Bellman-Ford algorithm.**

## I. Introduction

An internetwork consists of a collection of interconnected domains, where each domain is a collection of such resources as networks, routers, and hosts, under the control of a single administration. All the work in inter-domain and intra-domain routing has proceeded in two main directions: distance-vector protocols (e.g., BGP [19], IDRP [16], RIP [12], EIGRP [1]) in which routers exchange vectors of distances of preferred paths to known destinations, and link-state protocols (e.g., the inter-domain policy routing (IDPR) architecture [6], ISO IS-IS [15] and OSPF [21]) in which routers replicate topology information with which they compute their preferred paths.

The key advantage of protocols based on distance-vector algorithms (DVA) is that they scale well for a given combination of services taken into account in a cost metric. Because route computation is done distributedly, DVAs are ideal to support the aggregation of destinations to reduce communication, processing, or storage overhead. However, an inherent limitation of DVAs is that routers exchange information regarding path characteristics, not link or node characteristics. Because of this, the storage and communication requirements of *any* DVA grows proportionally to the number of combinations of service types or policies [17].

Because protocols based on link-state algorithms (LSA) replicate topology information at routers, they avoid the long-term looping problems of old distance-vector protocols (e.g., RIP). More importantly, an LSA exchanges information regarding link characteristics, which means that the complexity of storing and disseminating routing information to support multiple types of services and policies grows linearly with the service types and policies, not their combinations. However, today's LSAs use flooding to disseminate topology information to routers, which consumes excessive communication and processing resources, and communicating complete topology information to every router is unnecessary if a subset of links in the network is not used in the routes favored by some routers. To cope with the inherent overhead of flooding, today's LSAs organize the network or internet into areas connected by a backbone. However, this imposes additional network configuration problems and, as the results in [11] indicate, areas must be chosen carefully, together with the masks used to hide information regarding destinations in an area, for any performance improvement to be obtained with respect to a "flat" LSA. Furthermore, aggregating information in an LSA is much more difficult than in a DVA (e.g., see [8], [25]). Because LSAs require topology maps to be replicated at each router, different levels of topologies must be defined and routers must use multiple topology maps to aggregate information in an LSA (e.g., [21], [27]). In contrast, aggregating information in DVAs is very simple, because DVA exchanges information about distances to destinations, and such destinations can be a single network entity (e.g., router, network, host) or a group of entities (e.g., areas, confederations, clusters). Accordingly, the routing algorithm uses a single routing table with entries to individual or aggregated destinations [20].

Although the inherent limitations of LSAs and DVAs are well known, existing routing protocols or proposals for routing in large internets are based on these two approaches (e.g., see [4], [6]). This paper presents a new method for distributed, scalable routing in computer networks called *link vector algorithms*, or LVA. The basic idea of LVA consists of asking each router to report to its neighbors the characteristics of each of the links it uses to reach a destination through one or more preferred paths, and to report to its neighbors which links it has erased from its preferred paths. Using this information, each router constructs a *source graph* consisting of all the links it uses in the preferred paths to each destination. LVA ensures that the link-state information maintained at each router corresponds to the link constituency of the preferred paths used by routers in the network or internet. Each router runs a local path-selection algorithm or multiple algorithms on its topology table to compute its source graph with the preferred paths to each destination. Such path-selection

This work was supported in part by the Office of Naval Research (ONR) under Contract No. N-00014-92-J-1807 and by the Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0175. This work was presented in part at the SIGCOMM'94 Conference, London, UK, August 31 to September 2, 1994.

The authors are with the Baskin Center for Computer Engineering and Information Sciences, University of California, Santa Cruz, CA 95064, USA.

| Report Documentation Page | | | *Form Approved* *OMB No. 0704-0188* |
|---|---|---|---|
| Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. | | | |
| 1. REPORT DATE **SEP 1994** | 2. REPORT TYPE | | 3. DATES COVERED **00-09-1994 to 00-09-1994** |
| 4. TITLE AND SUBTITLE **Distributed, Scalable Routing Based on Vectors of Link States** | | | 5a. CONTRACT NUMBER |
| | | | 5b. GRANT NUMBER |
| | | | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | | | 5d. PROJECT NUMBER |
| | | | 5e. TASK NUMBER |
| | | | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064** | | | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | | | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | | | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |
| 12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited** | | | |
| 13. SUPPLEMENTARY NOTES | | | |
| 14. ABSTRACT | | | |
| 15. SUBJECT TERMS | | | |

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT **unclassified** | b. ABSTRACT **unclassified** | c. THIS PAGE **unclassified** | | **13** | |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std Z39-18

algorithm can choose any type of path (e.g., shortest path, maximum-capacity path, policy path) and the only two requirements for correct operation are for all routers to use deterministic algorithms that produce the same type of result when computing the preferred paths (e.g., one router can use Bellman-Ford also to compute shortest paths from its source graph, while others can use Dijkstra's algorithm) and that routers report all the links used in all preferred paths obtained using each.

Because LVAs propagate link-state information by diffusing link states selectively based on the distributed computation of preferred paths, LVAs reduce the communication overhead incurred in traditional LSAs, which rely on flooding of link states. Because LVAs exchange routing information that is related to link (and even node) characteristics, rather than path characteristics, this approach eliminates the complexity incurred with DVAs for routing under multiple constraints [17]. Regardless of the type of routing algorithm used, aggregation information (i.e., hierarchical routing) becomes a necessity to support routing in very large networks or internets. LVAs report links needed to reach destinations, not complete topology maps, and a destination can be a single entity or an aggregate of entities. Therefore, aggregation of information can take place in an LVA by adapting any of the area-based routing techniques proposed for DVAs in the past (e.g., [8], [18], [28]).

The following sections introduce the network model assumed throughout the rest of the paper, describe LVA, show that LVA converges to correct paths a finite time after the occurrence of an arbitrary sequence of link-cost or topological changes under the assumption that all routers run the same local algorithm(s) for the computation of preferred paths, calculate its complexity, and compare its average performance against the performance of an ideal LSA and the Distributed Bellman Ford (DBF) algorithm used in many DVAs.

## II. Link Vector Algorithm (LVA)

To describe LVA, an internet is modeled as an undirected connected graph $G = (V, E)$, where $V$ is the set of nodes and $E$ the set of edges. Routers are the nodes of the graph and networks or direct links between routers are the edges of the graph. Each point-to-point link in such a graph has two lengths or costs associated with it—one for each direction. Any point-to-point link of the graph exists in both directions at any one time. For a multi-point link, the cost of the link is assumed to be the same in all directions, and exists in all directions at any one time. An underlying protocol assures that

- A node detects within a finite time the existence of a new neighbor or the loss of connectivity with a neighbor.
- All messages transmitted over an operational link are received correctly and in the proper sequence within a finite time.
- All messages, changes in the cost of a link, link failures, and new-neighbor notifications are processed one at a time within a finite time and in the order in which they are detected.

Each router has a unique identifier, and link costs can vary in time but are always positive. Furthermore, routers are assumed to operate correctly, and information is assumed to be stored without errors. The same model can be applied to a single computer network.

The basic idea of LVA consists of asking each router to report to its neighbors the characteristics of every link it uses to reach a destination through a preferred path. The set of links used by a router in its preferred paths is called the *source graph* of the router. The topology known to a router consists of its adjacent links and the source graphs reported by its neighbors. The router uses this topology information to generate its own source graph using one or more local algorithms, which we call *path-selection algorithms.* A router derives a routing table specifying the successor, successors, or paths to each destination by running local algorithms on its source graph that can, of course, be part of the path-selection algorithms.

Figure 1(a) shows an example topology in which each link has the same cost in both directions. Figures 1 (b) through (e) show the selected topology known according to LVA with shortest-path routing at the routers highlighted in black. Solid lines represent the links that are part of the source graph of the respective router, dashed links represent links that are part of the router's topology table but not of its source graph. Arrowheads on links indicate the direction of the link stored in the router's topology table. A link with two arrowheads corresponds to two links in the topology table; since the source graph is a tree rooted at the black node in the case of shortest-path routing, only the direction pointing away from the black node can be part of the source graph. Router $x$'s source graph shown in Figure 1(b) is formed by the source graphs reported by its neighbors $y$ and $z$ (these are formed by the links in solid lines shown in Figures 1(c) and (d)) and the links for which router $x$ is the head node (namely links $(x, y)$ and $(x, z)$). A router's topology table may contain a link in only one direction (e.g., link $(y, u)$ in Figure 1(b)); this is because a router's source graph contains links only in the directions of its preferred paths.

In addition to the parameters of a link, the record of each link entry in the topology table contains the set of neighbors that reported the link, and control information used to detect the validity of updates received for that particular link.

The basic update unit in LVA is a link-state update reporting the characteristics of a link; an update message contains one or more updates. For a link between router $x$ and router or destination $y$, router $x$ is called the *head node* of the link in the $x$ to $y$ direction. For a multi-point link, a single head node is defined. The head node of a link is the only router that can report changes in the parameters of that link.

The main complexity in designing LVA stems from the fact that routers with different topology databases can generate long-term or even permanent routing loops if the information in those databases is inconsistent on a long-term

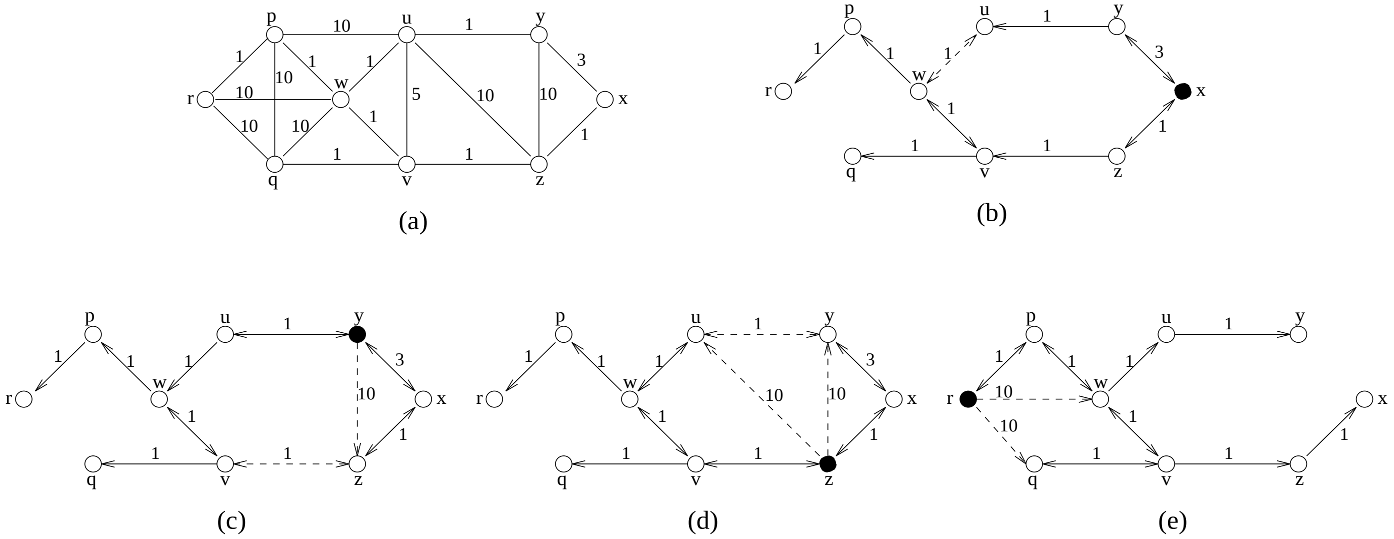

Fig. 1. Example topology. Solid lines indicate links in source graph; dashed lines indicate links in topology table but not in source graph; arrowheads indicate the stored direction.

or permanent basis.

Sending updates specifying only those links that a router currently uses in its preferred paths is not sufficient in LVA, because a given router sends incremental updates and may stop forwarding state information regarding one or more links that are not changing the values of their parameters. When this happens, it is not possible to ascertain if the router is still using those links in preferred paths if routers' updates specify only those links currently used in preferred paths. Simply aging link-state information would lead to unnecessary additional control traffic and routing loops, specially in very large internets. Therefore, to eliminate long-term or permanent routing loops, routers must not only tell its neighbor routers which links they use in their preferred paths, but also which links they no longer use. Accordingly, routers using LVA send update messages with two types of update entries: *add* updates and *delete* updates. An *add* update reports a link that has been added to the source graph of the sending router or whose information has been updated; a *delete* update specifies a link that has been deleted from the source graph of the sending router. An update specifies all the parameters of the link (just like in an LSA) and a router sends an update in a message only when a link is modified, added, or deleted in its source graph, not when the same unmodified link is used for a modified set of preferred paths; therefore, the number of update messages and the size of update messages do not necessarily increase with the number of paths, service types, or policies that a router uses.

A router reports its source graph to its neighbors incrementally; therefore, a typical update message in LVA contains only a few *add* and *delete* updates. Of course, when a router establishes a new link, it has to send its entire source graph to the new neighbor; this is equivalent to the LSA case in which a router sends its entire topology table to a new neighbor, or the DVA case in which a router sends its entire routing table to a newly found neighbor.

Because of delays in the routers and links of the internet, the *add* or *delete* updates sent by a router may propagate at different speeds along different paths. Therefore, a given router may receive an update from a neighbor with stale link-state information. The consistency of link-state information can be controlled on a link-by-link basis taking advantage of the fact that the only router that can change the information about a given link is its head node. More specifically, a distributed termination-detection mechanism is necessary for a router to ascertain when a given update is valid and avoid the possibility of updates circulating in the network forever [3], [2]. Termination-detection mechanisms based on sequence numbers similar to those used in a number of LSAs [3], [21], [22] or diffusing computations [9] can be used to validate updates.

A concrete embodiment of LVA for shortest-path routing, which we call LVA-SEN, is shown in Figures 2, 3, and 4. LVA-SEN determines the validity of updates using a sequence number for each link. The sequence number of a link consists of a counter that can be incremented only by the head node of the link. For convenience, a router is assumed to keep only one counter for all the links for which it is the head node, which simply means that the sequence number a router gives to a link for which it is the head node

| | |
|---|---|
| $(k, j)$ | link from $k$ to $j$ |
| $TT_i$ | topology table |
| $(k, j, l, ts, r)$ | entry in $TT_i$ for $(k, j)$ |
| $l$ | cost of the link |
| $ts$ | sequence number |
| $r$ | set of reporting nodes for link |
| $ST_i$ | source graph at router $i$ |
| $t$ | sequence number of router processing event |
| $N_i$ | set of neighbors of router $i$ |
| $(i, j, l, ts, type)$ | entry in update message |
| $type$ | add or delete operation |

Fig. 2. LVA-SEN Variables and Data Structures

```
procedure update ($i, n$, message)
    begin
        u_message = $\emptyset$
        updated = update_topology_table ($i$, message, u_message)
        if u_message $\neq \emptyset$
            send ($n$, u_message)
        end if
        u_message = $\emptyset$
        if updated then
            build_shortest_path_tree ($i, TT_i, NewST_i$)
            build routing table
            compare_source_graphs ($i, ST_i, NewST_i$, u_message)
            remove marked links from $TT_i$
            if u_message $\neq \emptyset$ then
                for all $x \in N_i$ do
                    send ($x$, u_message)
                end for
            end if
            $ST_i = NewST_i$
            $t = t + 1$
        end if
    end update

procedure link_change($i$)
    begin
        update($i, i, \{(i, j, l^i_j, t, \text{add})\}$)
    end link_change

procedure link_down($i, j$)
    begin
        message = $\emptyset$
        $N_i = N_i - \{i\}$ (but keep sequence number)
        for all $(k, m) \in TT_i$ do
            $TT_i(k, m).r = TT_i(k, m).r - \{j\}$
            if $TT_i(k, m).r = \emptyset$ or $TT_i(k, m).r = \{i\}$ then
                message = message $\cup \{(TT_i(k, m), \text{delete})\}$
            end if
        end for
        message = message $\cup \{(i, j, \infty, t, \text{delete})\}$)
        update (i, i, message)
    end link_down

procedure link_up($i, j$)
    begin
        $N_i = N_i \cup \{j\}$
        update ($i, i, \{(i, j, l^i_j, t, \text{add})\}$)
        u_message = $\emptyset$
        for all $(k, m) \in ST_i$ do
            u_message = u_message $\cup$ ($TT_i(k, m)$, add)
        end for
        send(j, u_message)
    end link_up

procedure node_up ($i$);
    begin
        $t = 0$
        message = $\emptyset$
        $N_i = \{x | \exists (i, x), l^i_x < \infty\}$
        for all $x \in N_i$ do
            $TT_i = TT_i \cup (i, x, l^i_x, t, \{i\})$
            $ST_i = ST_i \cup (i, x, t)$
        end for
        build new routing table
        for all $x \in N_i$ do
            send ($x$, query)
        end for
        answers_received = 0
        while answers_received < $|N_i|$ do
            receive (answer)
            $t = \max\{t, answer.t\}$
            update_topology_table ($i$, answer, u_message)
        end while
        $t = t + 1$
        for all $x \in N_i$ do
            $TT_i(i, x).t = t$
        end for
        build_shortest_path_tree($i, TT_i, NewST_i$)
        build new routing table
        $ST_i = \emptyset$
        u_message = $\emptyset$
        compare_source_graphs ($ST_i, NewST_i$, u_message)
        for all $x \in N_i$ do
            send ($x$, u_message)
        end for
        $ST_i = NewST_i$
        $t = t + 1$
    end node_up

procedure answer_query($i, j$)
    begin
        if $(i, j) \notin ST_i$ then
            $ST_i = ST_i \cup (i, j)$
            build routing table
        end if
        for all $(k, m) \in ST_i$
            u_message = u_message $\cup$ $TT_i(k, m)$
        end for
        u_message.t = $t_j$
        send ($j$, u_message)
    end answer_query
```

Fig. 3. LVA-SEN Specification

can be incremented by more than one each time the link parameters change values. The information regarding each link in a router's topology table is augmented to include the sequence number of the most recent update received, which was generated by the head node of that link.

For simplicity, the specification of LVA-SEN assumes that unbounded counters are used to keep track of sequence numbers and that each router remembers the sequence number of links deleted from its topology table long enough for the algorithm to work correctly. In practice, if finite sequence numbers are used to validate updates, a reset mechanism is needed to recycle sequence numbers. An age field serves this purpose [3]; when a link is deleted from the topology table, the router maintaining the table labels the link as deleted, which means that it is not used to compute new source graphs, and keeps the link entry until the age field of the entry expires. To limit the size of the age field, the head node of each link sends an *add* or *delete* update for that link periodically (depending on whether or not the link is in its source graph), even if no changes occur in the link. The maximum age of a link is then a multiple of the time it takes for an update to propagate throughout the network.

Procedures *update*, *update_topology_table*, and *compare_source_graphs* in Figure 3 are the core of LVA-SEN, in that these procedures are performed to update the data structures held at the router each time a router processes an input event (e.g., a message from one of its neighbors, or a link-cost change notification from an underlying protocol).

In LVA-SEN, when a router processes an *add* or *delete* update, it first compares the sequence number in that update against the sequence number maintained for the same link in the topology table. The update is processed if either it specifies a larger sequence number than the one stored in the topology table, or no entry for the link exists in the topology table. In the case of an *add* update, the link state is added to the topology table, or the new values of the link parameters replace the current entry, or the reporting node is added to the set of nodes that reported that link. For the case of a *delete* update, if there is an entry concerning the reported link in the topology table, then the reporting node

```
procedure update_topology_table ($i$, message, u_message)
  begin
    updated = false
    for all $m = (j, k, l, ts, type)$ do
      if $type$ = add then
        if $(j,k) \in TT_i$ then
          if $TT_i(j,k).t < m.ts$ then
            $TT_i(j,k) = m$
            $TT_i(j,k).r = \{$message.source $\}$
            updated = true
          else if $TT_i(j,k).t = m.ts$ and $i \neq$ message.source then
            $TT_i(j,k).r = TT_i(j,k).r \cup \{$ message.source $\}$
            updated = true
          end if
        else if ($i \neq j$ or message.source = $i$)
              and ($TT_i(j,k).t <= m.ts$) then
          $TT_i = TT_i \cup m$
          updated = true
        end if
        if $TT_i(j,k).t > m.ts$ then
          if $(j,k) \in ST_i$ then
            u_message = u_message $\cup$ ($TT_i(j,k)$, add)
          else
            u_message = u_message $\cup$ ($TT_i(j,k)$, delete)
          end if
        end if
      else if $type$ = delete
        if $TT_i(j,k).t < m.ts$ then
          if $(j,k) \in TT_i$ then
            mark $(j,k)$ as deleted
            updated = true
          else
            $TT_i(j,k).t = m.ts$
          end if
        else if $TT_i(j,k).t = m.ts$ then
          if $(j,k) \in TT_i$ then
            $TT_i(j,k).r = TT_i(j,k).r - \{$message.source$\}$
            if ($TT_i(j,k).r = \emptyset$ or $TT_i(j,k).r = \{i\}$)
                and $i \neq$ message.source then
              mark $(j,k)$ as deleted
              updated = true
            end if
          end if
        else if $TT_i(j,k).t > m.ts$ then
          if $(j,k) \in ST_i$ then
            u_message = u_message $\cup$ ($TT_i(j,k)$, add)
          else
            u_message = u_message $\cup$ ($TT_i(j,k)$, delete)
          end if
        end if
        if $TT_i(j,k).l = \infty$ and $TT_i(j,k).t < m.ts$ then
          $TT_i(j,k).t = m.ts$
        end if
      end if
      if $j$ = message.source and $j \in N_i$ then
        store sequence number of neighbor
      end if
    end for
    return updated
  end update_topology_table


procedure compare_source_graphs ($i, ST_i, NewST_i$, u_message)
  begin
    for all $(j,k) \in NewST_i$, $((j,k) \notin ST_i$
        or $NewST_i(j,k).ts > ST_i(j,k).ts)$ do
      u_message = u_message $\cup$ ($j, k, TT_i(j,k).ts, TT_i(j,k).l$, add)
    end for
    for all $(j,k) \in ST_i$, $(j,k) \notin NewST_i$ do
      if $i = j$ then
        u_message = u_message $\cup$ ($j, k, t, TT_i(j,k).l$, delete)
      else
        u_message = u_message
                $\cup$ ($j, k, TT_i(j,k).ts, TT_i(j,k).l$, delete)
      if
    end for
  end compare_source_graphs
```

Fig. 4. LVA-SEN Specification (Cont.)

is removed from the set of reporting nodes, and the link is deleted from the topology table if that set becomes empty and the link is not an outgoing link of finite length. An update is discarded if it specifies a sequence number that is smaller than the one in the topology table, in this case, the receiving node prepares an update for the same link intended to correct the information stored by its neighbor.

Dijkstra's algorithm [3], or any other shortest-path algorithm, is run on the updated topology table to construct a new source graph, which constitutes a shortest-path tree; in this case, the new routing table is generated together with the source graph. The router compares the new source graph against its old one (procedure *compare_source_graphs*), and an update message is sent with the differences between the two. In addition to changes in membership, a link in the source graph is considered different if its sequence number is changed. If the different link is in the new source graph, then an *add* update about this link is added to the update message. If the link is in the old source graph but is not in the new one, then a *delete* update is added. If any of the link entries refer to the state of an outgoing link of the node itself, then it gets a current sequence number. When a router sends an update message, it increments its sequence-number counter and discards its old source graph.

If a link cost changes, then its head node is notified by an underlying protocol. The node then runs *update* with the appropriate message as input. This holds for simple changes in link cost, as well as for a link failure. In the latter case, the link cost is set to infinity. The same approach is used for a new link or a link that comes up again after a failure. In the case of a failing node, all its neighbors are notified about the failure of their links to the failed node; these nodes then remove the failed node from the list of reporting nodes for all affected links, and therefore obtain an accurate picture of the topology after running the *update* procedure. Multiple changes in the status of nodes or links may be implied by the input event (e.g., a message received, a link failure). A node processes all such changes before it sends its own update message.

We assume that the node that comes up for the first time after a failure does not 'remember' any information that it previously had; in particular, it does not know the last sequence number it used. After initializing its data structures, the node that comes up sends a query to all its neighbors. In response, its neighbors send back their complete source graphs, plus the latest sequence number they received from the node (nodes store sequence numbers of neighboring nodes, which are updated when a link of some neighbor is changed). The node collects all this information, updates its topology table and sequence number, and then performs the same steps as in the procedure *update*.

Consider the topology of Figure 1 and assume that link $(y, z)$ fails. Nodes $y$ and $z$ process this failure and call procedure *link_down*; because neither node uses the failed link in its source graph, no update results from this failure. Consider now the case in which link $(x, y)$ fails. Nodes $x$ and $y$ call procedure *link_down*. Because link $(x, y)$ is used in the source graph of both nodes, they must send an update message. The message sent by node $y$ to its

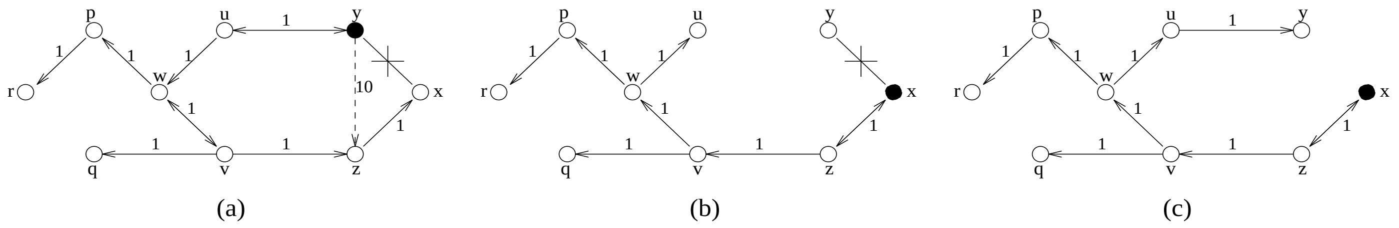

Fig. 5. View of topology at nodes $x$ and $y$ after link $(x, y)$ fails

neighbors specifies a *delete* update for link $(y, x)$ and *add* updates for links $(v, z)$ and $(z, x)$, which must now be used to reach nodes $z$ and $x$, respectively, and, therefore, must be added to node $y$'s source graph; furthermore, its update message also specifies delete updates for link $(x, z)$ which cannot be used to reach node $z$ anymore. Figure 5(a) shows node $y$'s view of the topology after the link failure. If node $u$ used the path $u, v, z, x$ to reach node $x$, the update from node $y$ causes no changes to $u$'s source graph and it simply updates its topology table based on $y$'s update.

On the other hand, when node $x$ processes the failure of link $(x, y)$, it has no path left to reach node $y$, because it has no link incident into node $y$ left in its topology table. Node $x$ sends a message containing *delete* updates for links $(x, y)$ and $(y, u)$, and an *add* update for link $(w, u)$. Figure 5(b) shows node $x$'s view of the topology at that time. When node $z$ processes node $x$'s update, it sends and update message that must contain an *add* update for link $(u, y)$ and a *delete* update for link $(x, y)$. When node $x$ processes that message, it obtains the view of the topology shown in Figure 5(c). Note that node $z$'s update does not create any changes in node $v$'s source graph, who must reach node $y$ through path $v, w, u, y$; therefore, node $v$ does not send any update as a result.

## III. Differences With Previous Methods

Three types of prior algorithms have been or can be used to compute preferred paths based on link-state information: link-state algorithms (LSA), *path-finding algorithms* (PFA), and *path-vector algorithms* (PVA).

### A. Differences with LSAs

LSAs are also called *topology broadcast algorithms.* In an LSA, information about the state of each link in the network is sent to every router by means of a reliable broadcast mechanism, and each router uses a local algorithm to compute preferred paths. The key difference between LSAs and LVAs is that each link-state update propagates to all routers in an LSA, while in LVA the update propagates to only those routers that use the corresponding link in a path to a destination and their neighbors. Therefore, the reliable broadcast mechanisms used in LSAs to ensure that all routers with a physical path to a source of link-state updates receives the most recent updates within a finite time (e.g., see [3], [9], [7], [13], [22]) are not directly applicable to an LVA. Furthermore, as argued before, a router using LVA must explicitly state which links it stops using.

Figure 1 helps to illustrate how LVA reduces storage and communication overhead compared to LSAs, even for the case of a fairly compact topology. An LSA would require each router to maintain a copy of the entire topology, with an entry for each link in each direction. Because a router's source graph contains the links in all its preferred paths, a router using LVA has the same number of paths available as with an LSA for any type of routing that applies the same constraints at every router (e.g., shortest path routing, maximum capacity routing). In the case that the type of routing permits different constraints to be applied at different routers (e.g., policy-based routing), LVA offers only a subset of the paths available with complete topology information. However, such a subset of paths is the same as that obtained with any PVA, which are used in standard inter-domain routing protocols.

In the worst case, each router's source graph contains all the links in the network and the LVA requires the same communication and storage overhead as an LSA. The number of updates and size of updates in an LVA are bounded by a number proportional to the number and size of updates in an LSA, because in that case update messages contain *add* updates reporting changes to the parameters of network links, just as in an LSA.

### B. Differences with PFAs

The basic idea in a PFA is for each router to maintain the shortest-path spanning tree reported by its neighbors (i.e., those routers connected to it through a direct link or a network), and to use this information, together with information regarding the cost of adjacent links, to generate its own shortest-path spanning tree. An update message exchanged among neighbors consist of a vector of entries that reports incremental or full updates to the sender's spanning tree; each update entry contains a destination identifier, a distance to the destination, and the second-to-last hop in the shortest path to the destination. Several PFAs have been proposed (e.g., see [5], [14], [23]). Another PFA by Riddle [26] is similar to the PFA method just mentioned in that a router communicates information regarding the second-to-last hop in the shortest path to each known destination. However, it uses exclusionary trees, rather than shortest-path spanning trees, and the cost of the link between the second-to-last hop and the destination, rather than the distance to the destination. An exclusionary tree sent from router $x$ to router $y$ consists of router $x$'s entire shortest-path tree, with the exception of the subtree portion that has node $y$ as its root. Riddle's algorithm does not use incremental updates.

Of course, any path-finding algorithm can use the cost of the link between the second-to-last hop and the destination, rather than the distance to the destination. However, the set of preferred paths used by a node to reach other nodes need not constitute a tree in LVA and it is always a tree in a path-finding algorithm. There are many reasons why routers may want to communicate link-state information of preferred paths that do not correspond to a tree. For example, if multiple shortest paths are desired, a router will communicate links along multiple preferred paths to each destination. Because there can be multiple links leading to the same node in the subgraph of preferred paths communicated by a router, a router that receives an incremental update from a neighbor cannot simply assume that the link from node $a$ to node $b$ communicated by its neighbor can substitute any previously reported link from another node $c$ to node $b$ by the same neighbor; therefore, the update mechanisms used in path-finding algorithms to update the subset of link states maintained at each router are not applicable to LVA.

### *C. Differences with PVAs*

With PVAs, routers exchange distance vectors whose entries specify complete path information for any destination they need to reach. The existing internet routing protocols based on PVAs (BGP [19] and IDRP [24]) do not exchange link-state information per se, but such information can be exchanged in a PVA by including it as part of the information for each hop of the reported path in an update or update entry. This, however, would become very inefficient when the size of the network and the number of link-state parameters are large, or when multiple preferred paths to each destination are desired.

LVAs provide routers with all the path information that PVAs provide, but with far less overhead. This is the case because a router that uses a given link in one or more preferred paths reports that link only once in LVA, while it has to include the link in each preferred path it reports using a PVA.

As we have noted, any routing algorithm that operates with less than a complete topology map reduces the set of all the possible paths that could be used to reach destinations when different constraints are applied at different nodes. In practice, this is not a problem, an LVA used for a given type of routing provides routers with exactly the same information than a PVA would for the same type of routing, because the routers' source graphs contain all the links in all the preferred paths selected by the routers. Therefore, any routing constraints or policies that can be supported with a PVA can be supported with an LVA, and practical routing protocols can be developed based on LVAs that provide the same functionality supported in BGP and IDRP, for example [24].

A more subtle difference between LVA and a PVA using link-state information is that routers using LVA determine whether or not an update to a link state is valid based on the timeliness of that update alone, just as in an LSA. In contrast, a router using a PVA that communicates link-state information still has to operate on a path-oriented basis, i.e., the timeliness of an update refers to an entire path, not its constituent links; therefore, even if a router is able to ascertain that a given update is more recent than another, that update may still use link-state information that is outdated (e.g., regarding links that are far away in the path). To eliminate the possibility of using stale link-state information in an adopted path, each link of the path could be validated (with a sequence number, for example), but this becomes inefficient in a large internet.

## IV. Correctness of LVA

Theorem 1 below shows that LVA is correct for multiple types of routing under the assumptions introduced in Section II and the additional assumptions that there is a finite number of link cost changes up to time $t_0$, that no more changes occur after that time, and that routers can correctly determine which updates are more recent than others. Corollary 1 then shows that routing tables do not contain any permanent loops. Verifying that finite sequence numbers and age fields can be used correctly to validate updates can be done in a manner similar to that used for finite sequence-numbering schemes used in LSAs [3].

Correctness for LVA means that, within a finite time after $t_0$, all routers obtain link-state information that allow them to compute loop-free paths that adhere to the constraints imposed by the local algorithms they use to compute preferred paths, and to forward packets incrementally.

Because our proof of correctness is intended for many different types of routing, not only shortest-path routing, we must specify what we mean by the correct operation of a path-selection algorithm. Consider the case in which each router in the network has complete and most recent topology information in its topology table and runs the same path-selection algorithm on it. In this case, it is evident that, for permanent loops to be avoided, the way in which the path-selection algorithm chooses routes must be deterministic. Assuming that the same deterministic path-selection algorithm is executed at each router using a complete and most recent copy of the topology, the preferred paths at any router for each destination constitute a directed acyclic graph (DAG). Furthermore, the union of the DAGs of any set of routers for the same destination in the network is also a DAG. Therefore, there are no permanent loops in the routing tables computed in this case.

*Definition 1:* A correct path-selection algorithm is one that produces the same loop-free paths when it is provided with the same complete and correct topology information.

It is not necessary for all routers to use the same correct path selection algorithm to compute preferred paths for a given type of routing. All that is needed is for all the path-selection algorithms used for the same type of routing to produce the same loop-free paths when they are provided the same topology information.

*Definition 2:* Two or more path-selection algorithms are compatible if they produce the same loop-free paths when they are provided with the same topology information.

As we have stated in the description of LVA, all routers use the same path-selection algorithm, or compatible path-selection algorithms, to compute the same type of preferred paths (e.g., shortest path, maximum capacity), and report all the links used in all the preferred paths obtained through all the path-selection algorithms. Therefore, the rest of this section can assume that a single path-selection algorithm is executed at every router, and that every router runs the same path-selection algorithm.

Because the topology tables of different routers running LVA need not have the same information, we cannot use the notion of having all topology tables containing the same information to ensure correct paths. The following definition specifies what a topology table should have for loop-free paths to be produced in LVA.

*Definition 3:* A router is said to have *consistent* link-state information in its topology table if it has the most recent link-state information regarding all the links for whom it is the head node, and the most recent link-state information corresponding to each of its neighbor's most recent source graph.

*Theorem 1:* A finite time after $t_0$, all routers have consistent link-state information in their topology tables and the preferred paths computed from those tables are correct.
**Proof:** Because the deterministic path-selection algorithm used at each router is assumed to be correct, all the proof needs to show is that

1. All routers eventually stop updating their topology and routing tables, and stop sending update messages to their neighbors.
2. All routers obtain consistent link-state information needed to compute correct preferred paths within a finite time after $t_0$.

These two properties are proven in the two lemmas included in the Appendix.

*Corollary 1:* The routing tables created by LVA do not contain any permanent loop.
**Proof:** Lemma 2 in the Appendix shows that the topology information at all routers is consistent within a finite amount of time after any change in link information. The topology information held at any router is a subset of the complete topology, and this subset contains all the information needed at this router to compute the correct preferred paths. Therefore, the preferred paths computed from any router's subset of the topology information must be a subset of the DAG computed in the case of each router having complete topology information. Any subset of a DAG is still a DAG; and the union of any such DAGs also forms a DAG, because that union is also a subset of the DAG obtained with complete topology information. Hence, the routing tables computed by LVA with a correct path-selection algorithm do not contain permanent loops.
**q.e.d.**

## V. Complexity of LVA

This section quantifies the communication complexity (i.e., number of messages needed in the worst case), time complexity (number of steps), computation complexity, and storage complexity [10] of LVA for shortest-path routing after a single link change.

### A. Communication Complexity

The number of messages per link cost change is bounded by twice the number of links in the network. To prove that this is the case, it suffices to show that any update can travel each link at most twice. Assume that an update concerning link $l$ arrives at some arbitrary node $n$ for the first time; there are two possibilities to consider:

1. The link is used in the source graph of $n$. If this is the case, the corresponding link-state information is sent to some neighbor $n_1$ over some link $l_1$. There are two possibilities at this router:
   (a) $n_1$ uses $l$: If the information was already known and used at $n_1$, then no further update will be sent over $l_1$ (or any other link adjacent to $n_1$). If it was not previously known at $n_1$, then an update will be sent to all neighbors of $n_1$, including one over $l_1$ to $n$. From $n$, no further update with information concerning $l$ will be sent over $l_1$, until newer information becomes available.
   (b) $n_1$ does not use $l$: $n_1$ will not sent any update with information concerning $l$, in particular none over $l_1$.
2. The link is not used at $n$, in which case no further update will be sent.

From the above, it follows that the number of messages is at most in the order of the number of links in the network ($O(|E|)$).

### B. Time Complexity

If the cost of links is not directly related to the delays incurred over such links, the number of steps required for any link change is $O(x)$, where $x$ is the number of nodes affected by the change. This can be shown by the following argument: the information about a changed link travels along all the shortest paths that contained the link before the change, and also along all shortest paths that will contain the link after the change. No other router than those along the paths and their neighbors will be notified about the change.

In the worst case, all the affected routers lie along one long path, thus causing $O(x)$ communication steps. In general, the paths on which the information is forwarded together with the affected routers form a directed, acyclic graph, and the upper bound for the steps required is given by the length of the longest simple path in that graph.

Because link failures and recoveries are handled as special cases of link cost changes, and router failures are perceived by the network as link failures for all their links, it is clear that $O(x)$ communication steps are also incurred in these cases. The case of a recovering node involves the nodes getting the complete source graphs from its neighbors, which takes no more steps than the number of neighbors, before the links of the routers again are handled as changing their cost to some finite value. Hence, the same upper bound of $O(x)$ applies.

This worst case is the same as the complexity of the best DVA [10]. On the other hand, if the link costs reflect the delay of the links, the complexity for LVA reduces to $O(d)$, where $d$ is the (delay) diameter of the network. The reasons for this are that the information travels along the shortest paths and a router receiving new information can trust the neighbor that reports the most recent link-state for the associated link; most importantly, the node will discard older information from other neighbors. Therefore, a router does not have to wait for link state updates to reach it through the slower paths, as is in the case in DVAs. The flooding technique used in LSAs also takes $O(d)$.

### C. Complexity of Computations at Routers

The most important routines to analyze are *update* and *update_topology_table*. Most other procedures just call *update* with the appropriate input message. One part of *update* is the shortest path finding algorithm (Dijkstra) with a complexity of $O(|V|^2)$. The routing table can then be computed in time $O(|V|)$, and the update message can be assembled using *compare_source_graphs* in less that $O(|V|^2)$ time.

The complexity of the main loop of procedure *update_topology_table* is determined by the size of the update message. In the worst case, this message could contain information about every link, resulting in running time $O(|E|) \leq O(|V|^2)$. This case seems highly unlikely, though.

In "normal" cases, we would expect an update message to contain information about some path plus possibly a second path that has to be deleted. A path can have at most length $|V|-1$, leading to an expected complexity of $O(|V|)$. The amount of work in the other loops is bounded by the number of nodes in the network.

All together, the overall worst case complexity for the procedure *update* is $O(|V|^2)$, mainly due to the shortest path algorithm. The corresponding complexity in a PVA is $O(|V|)$ for a single type of service.

Note that there also is the hidden complexity of accessing the topology table; this problem can be solved using a (dynamic) hash table, which has an expected constant access time.

### D. Storage Complexity

In the worst case, the topology table of each router maintains the whole topology, making the storage requirement $O(|V|^2)$. In addition, both the shortest path tree and the routing table require $O(|V|)$ storage, which is also the case for link state algorithms. Keeping track of reporting neighbors in LVA can be implemented by means of a bit vector. Because the identifiers of a router's neighbors can be stored in an ordered list, a single bit per neighbor suffices to indicate whether it is a member of the set of reporting nodes for a given link or not. Therefore, for each link in the topology table, the information about the reporting nodes adds only a constant amount of storage (which should be a few bytes for all practical purposes).

On the average, we expect the storage for the topology table to be by far smaller than $O(|V|^2)$. Because the goal is to keep as sparse a subset of the whole topology as possible (e.g., close to a tree), our conjecture is that the required storage space is closer to $O(|V|)$ for such simple types of routing as shortest-path routing. This seems realistic, even the small topology shown as example in Figure 1 revealed a significant saving of required space when compared to an algorithm that stores the complete topology at all routers. In contrast, the LSAs used today have to store the complete topology. Though the storage required for DVA is linear in the number of routers, routers have to store the routing tables of their neighbors. Therefore, DVAs' storage requirement really become $O(|V||N_k|)$ at router $k$, where $N_k$ is the set of neighbors of node $k$.

## VI. Simulation

We compare LVA-SEN, DBF, and an ideal LSA in terms of the number of steps and updates that are required for the algorithm to converge (i.e., the algorithm stops sending messages), and the size of these updates. When a node receives an update message, it compares its local step counter with the sender's counter, takes the maximum and increments the count. In all three algorithms, update messages are processed one at a time in the order in which they arrive. Both LVA-SEN and LSA use Dijkstra's algorithm to compute the local shortest-path tree. The results presented are based on simulations* for the DOE-ESNET topology [11]; similar results were obtained for other smaller topologies. The graphs show the results for every single link changing cost from 1.0 to 2.0 (Fig. 6, 7, 8), every link failing (Fig. 9, 10, 11) and recovering (Fig. 12, 13, 14), as well as every node failing (Fig. 15, 16, 17) and recovering again (Fig. 18, 19, 20). All changes were performed one at a time, and the algorithms had time to converge before the next change occurred. The ordinate of Figures 6 to 14 and Figures 15 to 20 represent identifiers of the links and the nodes, respectively, that are altered in the simulation. In Figures 6, 9, 12, 15, and 18, the data points show the number of update messages sent, in Figures 7, 10, 13, 16, and 19 they show the size of these updates, and in Figures 8, 11, 14, 17, and 20, they show the number of steps needed for convergence.

LSA shows almost constant behavior for all single link cost changes (Figures 6, 8) because the same link-state update must be sent to all routers. In contrast, DBF and LVA-SEN propagate updates to only those routers affected by the link-cost change; LVA-SEN is the most efficient of the three algorithms. Each update message contains one link state in LSA, and an average of 1.77 links in LVA-SEN. Figures 9 and 11 show similar behavior of the three algorithms for link failures, the exception being DBF suffering from 'counting to infinity' in some cases. In almost all cases, LVA-SEN needs fewer update messages and fewer steps than LSA; the average size of an LVA-SEN messages is 2.89 links.

When a failed link recovers, DBF is superior to both LVA-SEN and LSA. LSA exhibits the same behavior as

*We thank MIL-3 for donating us the OPNET modeling and simulation tools under their university program.

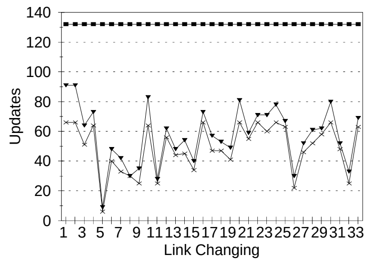

Fig. 6. Updates for link changes

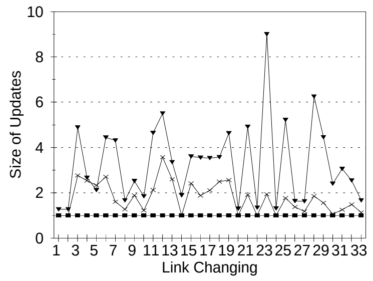

Fig. 7. Size of updates for link changes

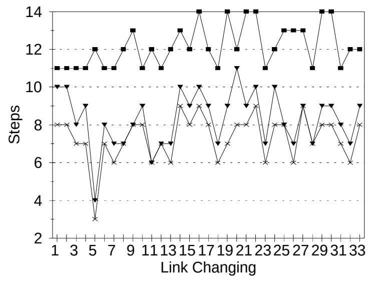

Fig. 8. Steps for link changes

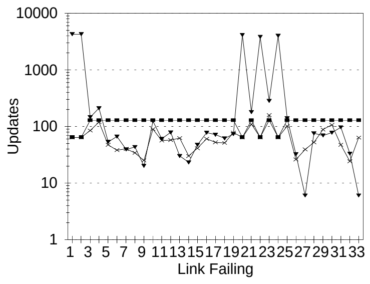

Fig. 9. Updates for link failures

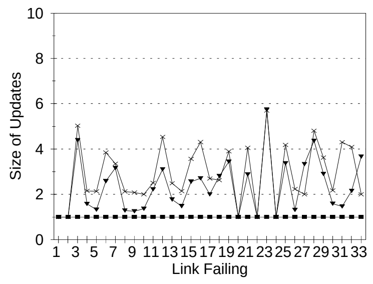

Fig. 10. Size of updates for link failures

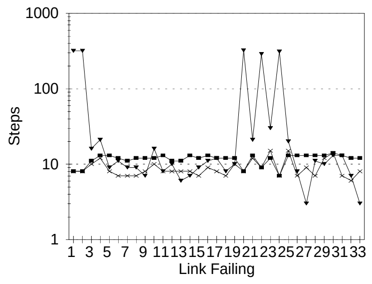

Fig. 11. Steps for link failures

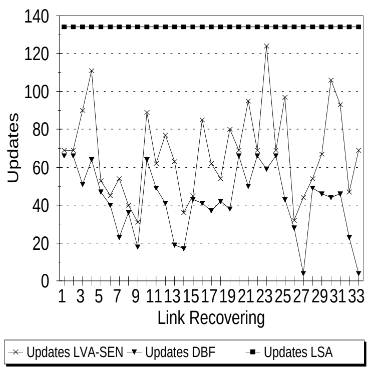

Fig. 12. Updates for link recoveries

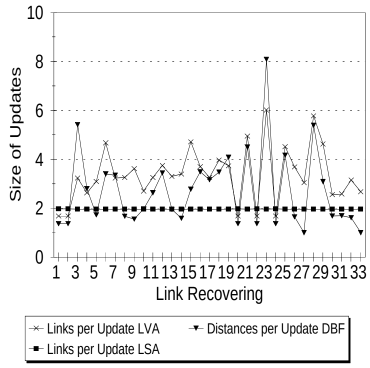

Fig. 13. Size of updates for link recoveries

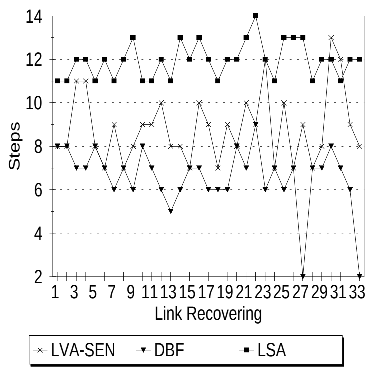

Fig. 14. Steps for link recoveries

with link-cost changes. With the exception of link 30, LVA-SEN is always better than LSA (Figures 12 and 14). The average LVA-SEN message is slightly more than three links; and LVA-SEN almost always requires less information to be sent than LSA and DBF, because messages in LSA are no longer one-link long due to the packets containing complete topology information sent over the recovering link.

For failing nodes, LSA usually has the best performance of the three algorithms. DBF always suffers from 'counting to infinity'. In almost all cases, LSA needs fewer steps and updates (Figures 15 and 17) than the other algorithms. The average message size for LSA is one link and 1.9 links for LVA-SEN.

DBF is superior to LSA when a node recovers, and LVA-SEN performs even better than DBF. LVA-SEN needs fewer steps and updates than the other algorithms (Figures 18 and 20). The average message size for LVA-SEN is 3.60 links), but of the three algorithms it still requires the least amount of information to be sent through the network.

Overall, the results of our simulations are quite encouraging. In terms of its overhead, LVA-SEN behaves much like DBF when link costs change and is always faster and produces less overhead traffic than LSA when resources are added to the network, and behaves much like the ideal LSA when links or routers fail. This is precisely the desired result, and indicates that LVAs are desirable even if multiple constraints are not an issue. It is apparent that LSA performs better than LVA only after node failures. The reason for this is that a failed node always impacts one preferred path for each node (i.e., every node's path to the failed node), which implies at least one *delete* update, and may also impact additional preferred paths of a subset of the nodes for other destinations, which may imply various *add* and *delete* updates for different links. Therefore, while LSA has to report only what happened to the links adjacent to the failed node, LVA needs to also add other links to bypass the failed node.

A simple way to improve the performance of an LVA after a node failure is the following: when a router detects a link failure or receives a *delete* update from a neighbor reporting a link failure, the router waits for a short hold-down time proportional to one propagation time between

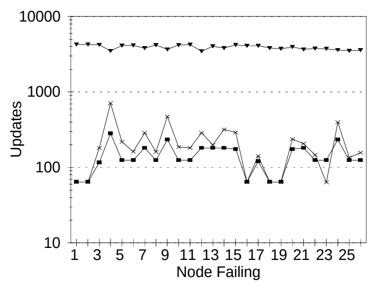

Fig. 15. Updates for node failures

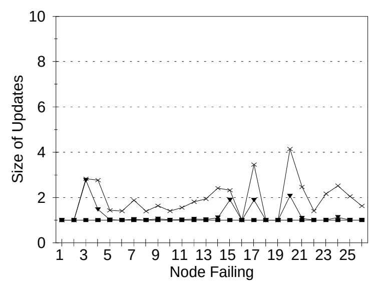

Fig. 16. Size of updates for node failures

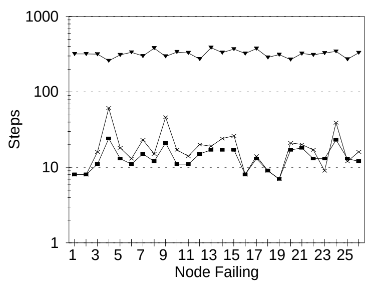

Fig. 17. Steps for node failures

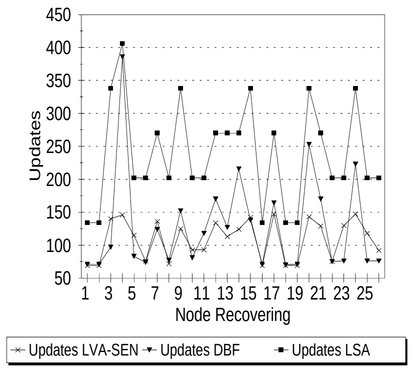

Fig. 18. Updates for node recoveries

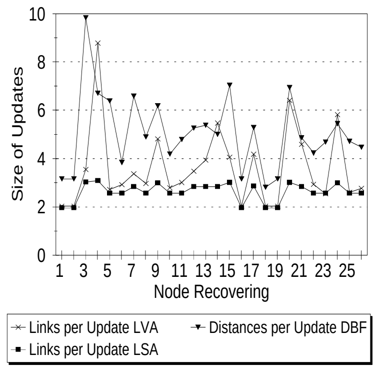

Fig. 19. Size of updates for node recoveries

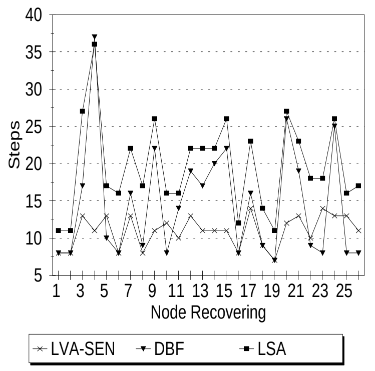

Fig. 20. Steps for node recoveries

neighbor routers to receive updates from other neighbors (which may contain *delete* updates for links adjacent to the same destination in the case of a node failure) before it updates its source graph and generates its own updates. Although the time a router takes to propagate updates due to link failures increases, the associated control traffic decreases; furthermore, routers away from the failed resource are more likely to get all the *delete* updates associated with a node failure in the same update message from any one of its neighbors, which means that the router will identify a destination that has failed and will not try to add links that no longer exist to its source graph to try to reach or use such destination as part of its preferred paths.

For larger topologies in which no hierarchical routing is used, we speculate that the relative differences in performance among the three algorithms are much the same. One possible exception may be that, as the networks become more richly connected any one link or node becomes less important to reach other nodes; therefore, with the exception of deleting links corresponding to the paths used to reach a failed node, other updates caused by a node failure will span only a few hops, because the probability of nodes not using a failed node to reach other destinations increases with network connectivity. For very large networks, however, hierarchical routing becomes a necessity; as we have stated, in this case LVAs can use simpler schemes than those needed in LSAs.

## VII. Conclusions

We have presented a new method for distributed routing in computer networks and internets using link-state information. LVAs enjoy nice scaling properties: like DVAs, LVAs scale well with the number of destinations by aggregating information; like LSAs, LVAs scale well with the number of service types because routers communicate link properties, not path properties in their updates. An important contribution of this paper is to show that LVA is correct under different types of routing, assuming that a correct mechanism is used for routers to ascertain which updates are recent or outdated.

LVAs open up a large number of interesting possibilities for routing protocols of large internets. LVAs can be used to develop intra-domain routing protocols that are based on link-state information but require no backbones or areas, and can take advantage of simple aggregation schemes developed for DVAs. LVA can also be applied to inter-domain routing protocols that provide the same functionality of BGP/IDRP while reducing the overhead incurred in communicating and storing updates.

## Appendix

## Proof of Theorem 1

*Lemma 1:* LVA terminates within a finite amount of time after $t_0$.

**Proof:** First note that there is a finite number of links in the network and that, by assumption, a finite number of link-state changes occur up to time $t_0$, after which no more changes occur. Also, by assumption, for each direction of a link whose parameters change, there is one router (the head node of the direction of the link) that must detect the change within a finite time; such a router updates its topology table and must then update its source graph. As a result of updating its source graph, the router can send at most one *add* update reporting the change in the state of the adjacent link, and at most one *add* or *delete* update for each of the links that have been added to or deleted from preferred paths as a result of the change in the adjacent link. Therefore, for any link $l_i$ in the network, its head node can generate at most one update for that link after time $t_0$.

A given router $x_1$ that never terminates LVA must generate an infinite number of *add* or *delete* updates after time $t_0$. It follows from the previous paragraph that this is possible only if $x_1$ sends such updates as a result of processing update messages from its neighbors; furthermore, because the network is finite, $x_1$ must generate an infinite number of updates for at least one link $l_1$. Because the network is finite, at least one of those neighbors (call it $x_2$) must send to $x_1$ an infinite number of update messages containing an update for either link $l_1$ or some other link $l_2$ that makes $x_1$ generate an update for link $l_1$. It follows from the previous paragraph and the fact that the network is finite that $x_2$ can send an infinite number of updates regarding link $l_1$ or $l_2$ to $x_1$ only if at least one of its neighbors (call it $x_3$) generates an infinite number of updates for either link $l_2$ or some other link $l_3$ that makes $x_2$ generate updates regarding link $l_1$ or $l_2$. Because the network is finite, it is impossible to continue with the same line of argument, given that the head node of any link can generate at most one update for that link after time $t_0$. Therefore, LVA can produce only a finite number of updates and update messages for a finite number of link-state changes and must stop within a finite time after $t_0$. **q.e.d.**

*Lemma 2:* All routers must have consistent link-state information in their topology databases within a finite time after $t_0$.

**Proof:** The definition of consistent link-state information at a router implies that the router knows all the links it needs to compute correct preferred path, and that the router has the most recent link-state information regarding all the links in its topology table. Proving that the router receives all the link-state information required to compute correct preferred paths can be done by induction on the number of hops $h$ of a preferred path. What needs to be shown is that the router knows all the links on that path within a finite time after $t_0$.

Consider some arbitrary preferred path from a router $i$ to some destination. For $h = 1$, the preferred path consists of one of router $i$'s outgoing links. Because of the basic assumption that some underlying protocol provides a router with correct information about its adjacent links within a finite time after the link-state information for such links changes, the lemma is true for this case. For $h > 1$, assume that the claim is true for any preferred path with fewer than $h$ hops.

Consider an arbitrary preferred path of length $h > 1$ from some router $i$ to a destination $j$. Let $k$ be router $i$'s successor on this path (i.e., the first intermediate router). Then, the subpath from $k$ to $j$ must have length $h-1$, and it must be one of router $k$'s preferred paths to $j$. Denote this path by $P_{kj}$. By the inductive hypothesis, router $k$ knows all the links on $P_{kj}$. Because router $i$ also knows (as in the base case) the most recent information about link $l_k^i$ within a finite time after $t_0$, it suffices to show that router $k$ indeed sends the link information in path $P_{kj}$ to its neighbor router $i$.

Assume that $P_{kj}$ is a new path for router $k$, then router $k$ must update its source graph. Because $P_{kj}$ is a new path for router $k$, the information in the updated source graph concerning $P_{kj}$ is different than the information in the old source graph. Therefore, router $k$ must include this information as *add* updates in the update message that it sends to its neighbors. Because router $i$ is one of those neighbors, it must receive from $k$ all the information on $P_{kj}$ within a finite time after $t_0$.

By assumption router $k$ can determine which link-state information is valid (i.e., up to date). Accordingly, if $P_{kj}$ is already one of router $k$'s preferred paths, but experiences a change in the information of some of its constituent links, then those links with updated link-state information will be considered different in the new source graph as compared to the old source graph. Therefore, router $k$ must send the updated link-state information in $P_{kj}$ to its neighbor $i$ in *add* updates.

The same inductive argument holds for link-state changes resulting in links being deleted from a preferred path. In this case, an intermediate router that decides that a link should no longer be used in any of its preferred paths sends a *delete* update, which is propagated just like an *add* update. This completes the first part of the proof.

Having shown that a router receives the most recent information about the links used in its source graph within a finite time after $t_0$, it remains to be shown that it also receives the most recent information about all the links that are in its topology table, but not part of the source graph of preferred paths. There are two possible cases to consider of links in a router's topology table that are not used in its source graph: an adjacent link to the router, or a non-adjacent link is in the source graph reported by some of the router's neighbor. In the first case, it is obvious that the lemma is true because of the basic assumption of some underlying protocol providing the node with correct information about adjacent links within a finite time. The second case follows almost immediately from the first part of this proof. Because every neighbor of the router sends the appropriate *add* or *delete* updates about links added to or deleted from its source own graph, it must be shown that each such neighbor obtains consistent information about changes in its source graph, which was shown to be the case in the first part of this proof. **q.e.d.**

## References

[1] R. Albrightson, J.J. Garcia-Luna-Aceves, and J. Boyle, "EIGRP–A Fast Routing Protocol Based on Distance Vectors" *Proc. Networld/Interop 94*, Las Vegas, Nevada, May 1994.

[2] J. Behrens and J.J. Garcia-Luna-Aceves, "Distributed, Scalable Routing based on Link-State Vectors," *Proc. ACM SIGCOMM 94*, London, UK, August 1994.

[3] D. Bertsekas and R. Gallager, *Data Networks*, Second Edition, Prentice-Hall, Inc., 1992.

[4] I. Castineyra, J. N. Chiappa, and M. Steenstrup, *The Nimrod Routing Architecture*, Internet Draft, January 1995

[5] C. Cheng, R. Riley, S. Kumar, and J.J. Garcia-Luna-Aceves, "A Loop-Free Extended Bellman-Ford Routing Protocol without Bouncing Effect," *ACM Computer Comm. Review*, vol. 19, no. 4, pp. 224-236, September 1989.

[6] D. Estrin, M. Steenstrup, and G. Tsudik, "A Protocol for Route Establishment and Packet Forwarding across Multidomain Internets," *IEEE/ACM Trans. Networking*, vol. 1, no. 1, pp. 56-70, February 1993.

[7] E. Gafni, "Generalized Scheme for Topology-Update in Dynamic Networks," *Lecture Notes in Computer Science* (G. Goos and J. Hartmanis, Eds.), no. 312, pp. 187-196, 1987.

[8] J.J. Garcia-Luna-Aceves, "Routing Management in Very Large-Scale Networks," *Future Generation Computing Systems (FGCS)*, North-Holland, vol. 4, no. 2, pp. 81-93, 1988.

[9] —, " Reliable Broadcast of Routing Information Using Diffusing Computations," *Proc. IEEE Globecom 92*, Orlando, Florida, December 1992.

[10] —, "Loop-Free Routing Using Diffusing Computations," *IEEE/ACM Trans. Networking*, vol. 1, no. 1, 1993.

[11] J.J. Garcia-Luna-Aceves and W.T. Zaumen, "Area-Based, Loop-Free Internet Routing," *Proc. IEEE INFOCOM 94,* Toronto, Canada, June 1994.

[12] C. Hedrick, "Routing Information Protocol," RFC 1058, Network Information Center, SRI International, Menlo Park, CA, June 1988.

[13] P.A. Humblet and S.R. Soloway, "Topology Broadcast Algorithms," *Computer Networks and ISDN Systems*, vol. 16, pp. 179-186, 1989.

[14] P. Humblet, "Another Adaptive Distributed Shortest Path Algorithm," *IEEE Trans. Comm.*, vol. 39, no. 6, pp. 995-1003, June 1991.

[15] International Standards Organization, "Intra-Domain IS-IS Routing Protocol," ISO/IEC JTC1/SC6 WG2 N323, September 1989.

[16] International Standards Organization, "Protocol for Exchange of Inter-domain Routing Information among Intermediate Systems to Support Forwarding of ISO 8473 PDUs," ISO/IEC/JTC1/SC6 CD10747.

[17] J.M. Jaffe, "Algorithms for Finding Paths with Multiple Constraints," *Networks*, vol. 14, pp. 95-116, 1984.

[18] L. Kleinrock and F. Kamoun, "Hierarchical Routing for Large Networks: Performance Evaluation and Optimization," *Computer Networks*, vol. 1, pp. 155-174, 1977.

[19] K. Lougheed and Y. Rekhter, " Border Gateway Protocol 3 (BGP-3)," RFC 1267, SRI International, Menlo Park, CA, October 1991.

[20] J. McQuillan, "Adaptive Routing Algorithms for Distributed Computer Networks," BBN Rep. 2831, Bolt Beranek and Newman Inc., Cambridge MA, May 1974.

[21] J. Moy, "OSPF Version 2," RFC 1583, Network Working Group, March 1994

[22] R. Perlman, "Fault-Tolerant Broadcast of Routing Information," in *Computer Networks*, North-Holland, vol. 7, pp. 395-405, 1983.

[23] B. Rajagopalan and M. Faiman, "A Responsive Distributed Shortest-Path Routing Algorithm within Autonomous Systems," *Internetworking: Research and Experience*, vol. 2, no. 1, pp. 51-69, March 1991.

[24] Y. Rekhter, "Inter-Domain Routing Protocol (IDRP)," *Internetworking: Research and Experience*, Wiley, vol. 4, no. 2, pp. 61-80, June 1993.

[25] Y. Rekhter, S. Hotz, and D. Estrin, "Constraints on Forming Clusters with Link-State Hop-by-Hop Routing," Technical Report USC-CS-93-536, University of Southern California, Los Angeles, 1994

[26] G.G. Riddle, "Message Routing in a Computer Network," U.S. Patent assigned to AT&T Bell Telephone Laboratories, Inc., Patent Number 4,466,060, August 1984.

[27] W.T. Tsai, C.V. Ramamoorthy, W.K. Tsai, and O. Nishiguchi, "An Adaptive Hierarchical Routing Protocol", *IEEE Trans. Computers*, vol. 38, no. 8, pp. 1059–1075, 1989.

[28] P. Tsuchiya, "The Landmark Hierarchy: A New Hierarchy for Routing in Very Large Networks," *Computer Comm. Review*, vol. 18, no. 4, pp. 43-54, 1988.